The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

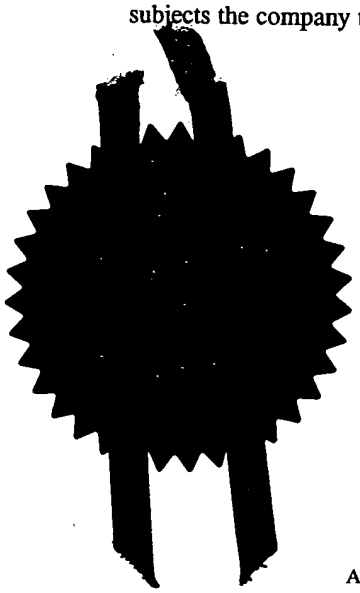
CERTIFIED COPY OF PRIORITY DOCUMENT

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.



Signed

Dated 20 February 2002

Patents Form 1/77

Patents Act 1977
(Rule 16)

The Patent Office

THE PATENT OFFICE
E
12 APR 2001
LONDON

D Y & CO

Request for a grant of a patent

(See the notes on the back of this form you can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference — P011318GB — 12 APR 2001

2. Patent application numbe
(The Patent Office will fill in this p
0109282.4

12APR01 E621991-3 D0224
P01/7700 0.00-0109282.4

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

ARM Limited
110 Fulbourn Road
Cherry Hinton
Cambridge
CB1 9NJ
United Kingdom

Patents ADP number *(if you know it)* 7498124002

If the applicant is a corporate body, give the country/state of its incorporation — United Kingdom

4. Title of the invention — RE-USEABLE HARDWARE/SOFTWARE CO-VERIFICATION OF IP BLOCKS

5. Name of your agent *(if you have one)* — **D YOUNG & CO**

"Address for service" in the United Kingdom to which all correspondence should be sent (including the postcode)

21 NEW FETTER LANE
LONDON
EC4A 1DA

Patents ADP number *(if you know it)* — **59006**

6. If you are declaring priority from one or more earlier patent applications, give the country and date of filing of the or each of these earlier applications and *(if you know it)* the or each application number

	Country	Priority application number *(if you know it)*	Date of filing *(day/month/year)*
1st			
2nd			
3rd			

7. If this application is divided or otherwise derived from an earlier UK application, give the number and filing date of the earlier application

Number of earlier application	Date of filing (day/month/year)

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*
a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

Yes

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	None
Description	4
Claim(s)	0
Abstract	0
Drawing(s)	0

10. If you are also filing any of the following, state how many against each item

Priority Documents	0
Translation of Priority Documents	0
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	0
Request for preliminary examination and search *(Patents Form 9/77)*	0
Request for substantive examination *(Patents Form 10/77)*	0
Any other documents *(Please specify)*	0

11. I/We request the grant of a Patent on the basis of this application.

Signature D Young & Co — Date

D YOUNG & CO
Agents for the Applicants

12 Apr 2001

12. Name and daytime telephone number of person to contact in the United Kingdom — David Horner — 023 80719500

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) If you need help to fill in this form or you have any questions, please contact the Patent Office on 01645 500505.

b) Write your answers in capital letters using black ink or you may type them.

c) if there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s) Any continuation sheets should be attached to this form.

d) If you answered 'Yes' Patents Form 7/77 will need to be filed.

e) Once you have filled in the form you must remember to sign and date it.

f) For details of the fee and ways to pay please contact the Patent Office.

Re-useable Hardware/Software Co-verification of IP Blocks

Alistair Bruce, John Goodenough

Abstract— **A novel use of remote procedure call techniques gives access to the software driver API of an IP block from a high-level verification testbench. This allows an IP block to be validated through its software interface as well as its hardware interface. Such a unified methodology has immediate benefits for IP block development. More importantly, it allows re-useable, system level verification components to be produced, which greatly enhance the value of an IP block.**

A combination of commercial testbench and co-simulation tools is used to demonstrate this methodology.

Index Terms— **Co-verification, IP block, remote procedure call, re-usability, testbench.**

I. Introduction

It is widely recognized that verification can take up to 70% of the development effort for a SoC design. This includes software as well as hardware verification [1].

Most hardware verification is done by simulation, however, until recently, software verification has had to wait for the hardware design to finish and be synthesized onto an FPGA. Then the software could be run on a prototype development board but this wait inevitably increased the development time.

One solution is to simulate the software and the hardware parts together using the commercial co-simulators that are now available. The benefits of this approach have been demonstrated elsewhere [2][3]. One notable benefit is the quality of the software component is much improved as the co-verification has eliminated errors that would otherwise only be found during system integration.

IP blocks are self-contained designs for common functions that can be re-used in a SoC, saving time and effort for the main function. During their development, verification can be done in the same way as for a full SoC and with the same benefits. However, an IP block is intended for re-use and so its verification should also be re-useable. In fact, an IP block is only viable if it takes less effort to integrate into a system than it would to develop the block from scratch.

Under these circumstances it is to be expected that nearly all the integration effort will be in verification. It has been suggested that verification represents up to 90% of the effort to integrate an IP block into a system [1].

An IP block should therefore be considered as having 3 components: hardware, software *and* verification. Re-useable verification is an important part of an IP block as it enhances its value by reducing the system integration effort. In other words, the value of the hardware and software are the reason for using an IP block but the verification component makes their re-use possible.

From this, it follows that the verification supplied with an IP block should be aimed at the needs of integration, not just development. Because IP blocks tend to be used as supplied, with no changes apart from those required by integration, functional verification of the block is less important. That has already been done during its development. Rather, the verification should be designed to show that the rest of the system correctly supports the IP block and that its presence does not upset the other parts of the design. This is very different from the standalone verification often supplied to verify synthesis of a soft IP block. Now the verification component supplies a kit of verification parts that can be used to build a set of verification tests as part of a system validation plan.

The latest verification techniques make use of high-level verification languages (HVL) supported by tools such as Veristy's Specman Elite or Sysnopsis' Vera. These have always provided good links to hardware simulation environments. Their purpose built test vector generation and coverage analysis tools make verification much easier and more thorough.

However, until recently these tools have not given equivalent facilities for embedded software running on SoC designs. These facilities are starting to become available and new verification strategies have become possible.

Of special interest is the verification of an IP block through the API of its software driver. The reason for this is that typically, at the system level, software rather than hardware controls an IP block. Whilst the high-level verification tools do not provided this directly, it has been possible to achieve it through a novel use of remote procedure call techniques.

The work described in this paper is part of the Systems Solution Methodology [4] being developed within ARM to support the development and deployment of ARM IP. As an example, the SmartCard Interface PrimeCell[1] was used to demonstrate the use of commercial EDA tools to implement a part of this methodology.

II. Verification Structure

Fig. 1 shows the proposed verification structure for an IP block. The symmetry emphasizes the equal status of the hardware and software components. Notice that the communication between the software and hardware components is considered to be part of the IP block and as

[1] The PrimeCell Peripherals are a range of AMBA-compliant IP blocks developed by ARM to facilitate SoC integration.

such is not directly driven by the testbench. Instead, it is checked by an interconnection-fabric protocol checker [5].

The two interfaces to the IP block that are driven by the testbench are the software interface of the driver API and the non-fabric hardware interface to other parts of the system.

A test scenario manager generates stimuli and collects responses from these two interfaces and uses scoreboarding to create self-checking, system level tests.

Both the hardware and software components of the IP block are checked for coverage. This is an important part of the methodology as it allows the user of the IP verification to quantify the coverage of the tests. This, in turn, means that the system validation can be checked for its coverage of a given IP block.

The internal state of the hardware and software components is monitored to measure coverage. The test scenarios can also use the state to verify correct internal behavior during the tests.

The hardware interface is accessed from the test environment via a bus functional model (BFM). This allows the tests to be written at a higher level of modeling, leaving the BFM to add the fine details.

The software interface is accessed from the test environment through the driver's API. This is in contrast to other methodologies where test programs have to be written and run on the co-simulation model in order to exercise the driver.

All the verification components have documented verification interfaces that allow them to be re-used in system tests without needing to know their internal working. For example, the system validation plan may call for a particular test to be performed with a FIFO both full and empty. The verification API and the coverage analysis supplied with the IP block give the appropriate system level facilities to implement this.

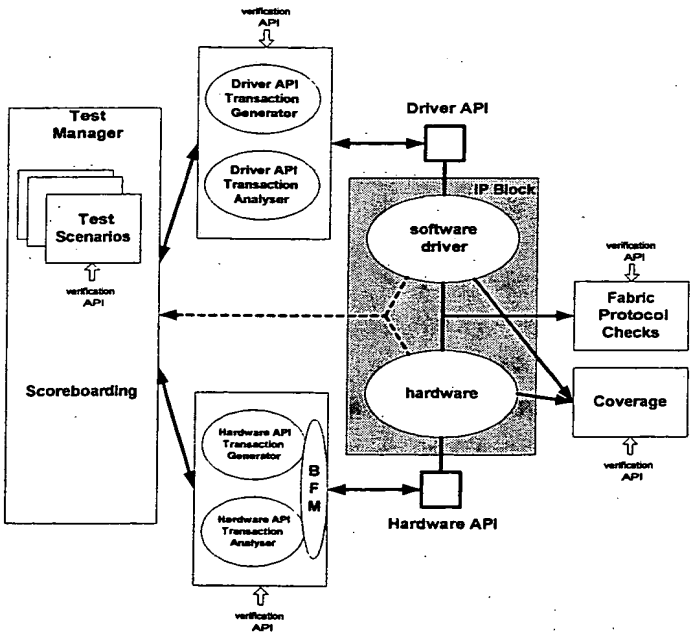


Fig. 1. Verification structure

III. Verification Access Of The Driver API

An important part of this methodology is the ability to call driver routines from the verification environment. This has been achieved by setting up a remote procedure calling mechanism [6][7] between the software running on the ARM co-verification model and the testbench running on the verification simulator.

The link between Specman and Seamless gives the ability to read from and write to global variables in the software simulation. It is also possible for methods in the testbench to wait for software variables to change value. This is enough to implement a single threaded remote procedure call (RPC) from an *e* language testbench to software routines running on the simulator.

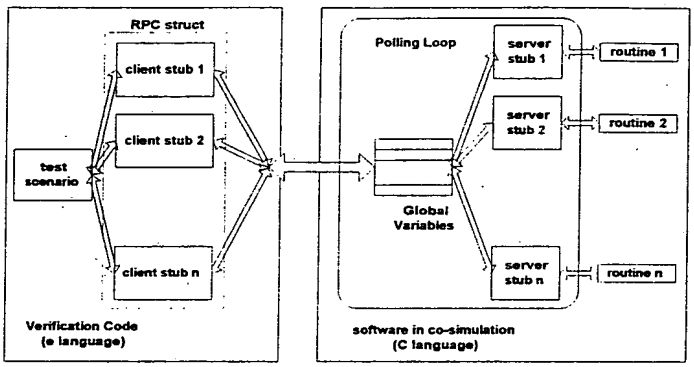


Fig. 2. RPC between verification code and software simulation

A simple client-server stub implementation is used (Fig. 2). On the client side, an *e* language struct is declared that encapsulates the client RPC functionality. The struct's init routine is extended to establish the connection with Seamless while the client stubs are declared as time consuming methods of the struct. Each client stub marshals the parameters and transfers them, along with a routine identifier, to a set of global variables in the software simulation. A further global variable is used to signal the start of the software routine. The client stub then waits for the software routine to signal its completion by resetting the start global variable (see Fig. 3).

```
apSCI_ParamsGet(oId          : apOS_SCI_oId,
                SetupParams : apSCI_sSetupParams) @clk is {
   'sw>SCI_command' = 2;
   'sw>SCI_command_p0' = oId;
   'sw>SCI_command_p1' = SetupParams.addr;
   'sw>SCI_start_command' = 1;
   wait until true('sw>SCI_start_command' == 0);

   var SetupBits : list of bit;
   SetupBits = 'sw>SetupData';
   unpack(NULL,SetupBits,SetupParams);
};
```

Fig. 3. Client stub in *e* language

The server side is implemented as a simple polling loop. Just before the loop is entered a global variable is set to signal that the server is ready. This allows the testbench to synchronize with the SoC co-simulation so that the hardware

reset can complete along with any bootstrap activity.

The polling loop checks for the "start" global variable to be set (by the client stub) indicating that a routine is to be called. When it is set, a switch statement selects the routine to call with the parameters already cached in global variables. When the routine completes the start global variable is reset and the polling loop resumes (see Fig. 4).

```
while(1)
{
   if(SCI_start_command)
   {
      switch(SCI_command)
      {
         .
         .
        case SCI_ParamsGet:
            apSCI_ParamsGet((apOS_SCI_oId)          SCI_command_p0,
                            (apSCI_sSetupParams *) SCI_command_p1);
         break;
         .
         .
         default:
         break;
      }
     SCI_start_command = 0;
   }
}
```

Fig. 4. Server polling loop

The mapping between *e* and the global variables has to be considered. The default transfer size is 32 bits so for types that fit into a single 32-bit word, such as int, bool and char, the default mapping is correct. The connection offers the ability to change the endianess if required.

Multi-word types are transferred as a list of bit and then unpacked into an *e* structure. In these cases the *e* definitions must be padded to the correct number of bits to match the alignment in the software simulation.

API Routines that have pointer parameters are handled specially. Matching data areas are set up on the client and server. The client transfers the server data across, modifies it as required and then transfers it back again. The routine can thus be called from the server stub with the pointer to the server data area.

Very often, a driver API makes use of call back routine parameters. This implies a call from the software simulation back into the into the verification code. These are handled by passing the address of a client stub routine on the server. This then uses the same RPC technique to call a routine via a server stub in the verification code.

Once the driver is accessible from the high-level verification language it is possible to write the sort of randomized test that has proven so useful in the hardware context. For example, it is now easy to generate a set of random parameter values for a routine and check its result.

The code in Fig. 5 illustrates both a multi-word parameter and the use of the HVL facilities to randomly exercise a driver routine. In this example the input and output are just printed for inspection. Normally the result would be checked against the expected value to form a self-checking test.

```
for i from 0 to 7 {
   gen SetupData keeping {
      .ClockFreq      in [10000000 .. 50000000];
      .DebounceTime   in [50 .. 200];
      .ActEventTime   in [30000 .. 60000];
      .DeactEventTime in [200 .. 400];
      .ATRStartTime   in [10000 .. 100000];
      .ATRDuration    in [10000 .. 100000];
      .CharacterTime  in [1200 .. 10000];
      .CardFreq       in [500000 .. 10000000];
   };
   'sw>SetupData' = pack(NULL,SetupData);
   SetupData.print();

   ErrCode = apSCI_ParamsSet(apOS_SCI_0,
                             SCI_ATRBuffer_addr,
                             TestCallback_addr,
                             SetupData_addr);
   out("ErrCode ",ErrCode);
   out("----------------------------------------");
};
```

Fig. 5. Testbench access of an API

IV. Implementation

The co-simulation environment was supplied by Mentor Seamless CVE, which contains an ARM co-verification model, and by Mentor ModelSim VHDL simulator. Verisity Specman Elite supplied the verification environment with links to both the HDL simulation and the software running on the ARM co-verification model. The link to the software has only recently become commercially available in Specman Elite version 3.3 and Seamless CVE version 4.0. It is a key enabling technology for this methodology.

The mapping of the verification structure onto these tools is shown in Fig. 6. Seamless supports the simulators to animate the software and hardware components of the IP block. The software is run on the cycle accurate instruction set simulator (an ARM co-verification model) whilst the ModelSim VHDL simulator animates the hardware component. Communication between the two simulators is mediated by the Seamless CVE co-simulation kernel and a bus interface model.

The verification components and test manager are implemented in the *e* language supported by Specman Elite.

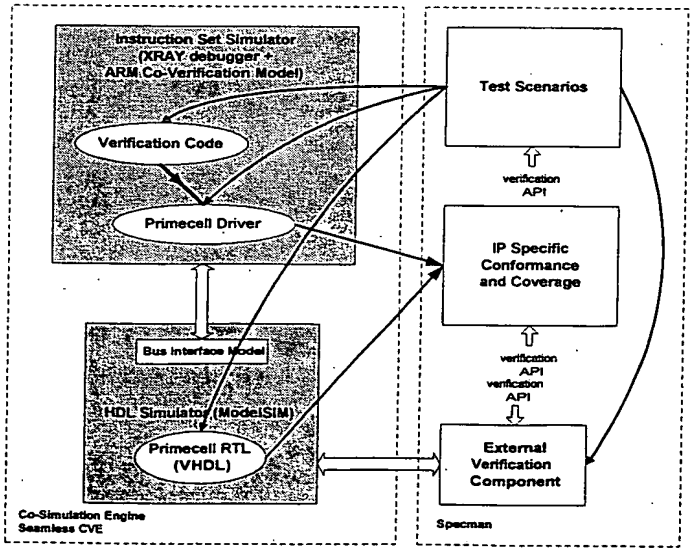


Fig. 6. Co-verification tool structure

The SmartCard Interface PrimeCell was incorporated into a typical SoC structure shown in Fig. 7. This was based on the AHB/APB VHDL testbench supplied as part of MicroPack 2.0[2].

For the SmartCard Interface example a smartcard bus functional model was written to give verification access to the hardware API.

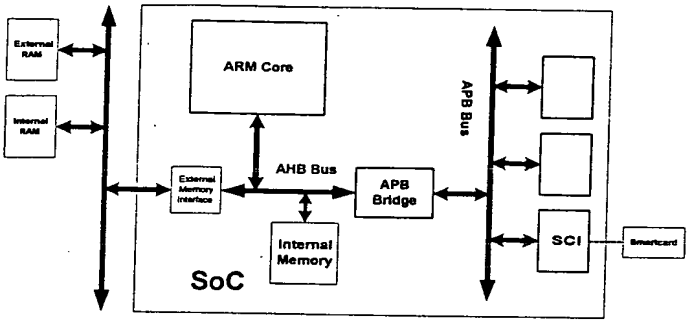


Fig. 7. SoC structure

V. Conclusion

The verification environment provided by Specman, Seamless CVE and ModelSim forms the basis of a complete software/hardware co-verification solution. The RPC technique described adds the ability to stimulate, monitor and check an IP block through its software interface. This is an important advance that allows IP blocks to be supplied with re-useable system verification components.

With this technique, the software is tested in the context of the hardware that it drives using the high-level verification generation and coverage facilities of the testbench. Furthermore, because the software and hardware models are accessible at the same time, it is possible to set up tests that would be impractical in other environments.

SoC design benefits from the re-useable conformance and checking components that are developed during the IP block design. These components are re-used to check that system integration has been successfully achieved.

The development of the software component of an IP block benefits from the earlier access to a complete validation environment with a shorter, more accurate and faster implementation.

Legacy IP blocks can benefit retrospectively from this methodology through the addition of validation components that bring them up to the same standard as new IP, thus easing their re-use in future SoC designs.

References

[1] Mark Peryer "Re-using Virtual Component functional verification in a SoC environment" IP 2000 Europe Tutorial

[2] Johann Notbauer, Thomas Albrecht, Georg Nierdrist, Stefan Rohringer, "Verification and Management of a multimillion-gate embedded core design" 36th ACM/IEEE Design Automation Conference, 1999 p.425.

[3] H. Nonoshita, "Hardware/Software Co-verification in a System LSI Design" Integrated System Design Magazine, May 2000.

[4] SSM Paper

[5] ACT Paper

[6] B.J. Nelson "Remote Procedure Call" XEROX PARC CSL-81-9, May 1981.

[7] Andrew D. Birrell, Bruce Jay Nelson, "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems Vol. 2, No. 1, February 1984 pp. 39-59.

[2] MicroPack is a versatile toolkit aimed at enabling the successful creation of AMBA-based components and SoC designs